Factorial Representations to Generate Arbitrary Search Distributions

Marc Toussaint
Institute for Adaptive and Neural Computation
University of Edinburgh, 5 Forrest Hill
Edinburgh EH1 2QL, Scotland, UK
mtoussai@inf.ed.ac.uk

ABSTRACT

A powerful approach to search is to try to learn a distribution of good solutions (in particular of the dependencies between their variables) and use this distribution as a basis to sample new search points. Existing algorithms learn the search distribution directly on the given problem representation. We ask how search distributions can be modeled indirectly by a proper choice of factorial genetic code. For instance, instead of learning a direct probabilistic model of the dependencies between variables (like BOA does), one can alternatively learn a genetic representation of solutions on which these dependencies vanish. We consider questions like: Can every distribution be induced indirectly by a proper factorial representation? How can such representations be constructed from data? Are specific generative representations, like grammars or L-systems, universal w.r.t. inducing arbitrary distributions? We will consider latent variable probabilistic models as a framework to address such questions and thereby also establish relations to machine learning concepts like ICA.

Categories and Subject Descriptors

F.2 [**Analysis of algorithms and problem complexity**]: General

General Terms

Algorithms, design, theory

Keywords

Search, probabilistic models, representations, genetic code

1. INTRODUCTION

Searching for a solution to a problem becomes complex when many variables of the solution interact. Estimation-of-Distribution Algorithms (EDAs, or Probabilistic model-building Genetic Algorithms [9]) are one approach to address such problems: One tries to learn a model of the distribution of previously found good samples. Learning such a model also implies to analyze the dependencies between variables. The model is then used to sample new search points.

Existing EDAs focus on learning probabilistic models directly on the given problem representation. If on this representation the dependencies between variables are complex, complex probabilistic models (like dependency trees or graphical models) have to be found.

From a biological perspective, one may argue that such search schemes can hardly be interpreted as an abstraction of an evolutionary processes based on independent mutational incidents—and thus do not directly help to understand how evolution was able to find complex solutions. From the optimization point of view, one may ask whether these methods could in principle scale up to situations where solutions are formed of millions of interacting variables, even when the interactions are actually not too complex because they follow certain patterns (cf. [4]).

The approach to (re-)represent solutions in such a way that dependencies between variables vanish can be regarded as an antipode to the approach to learn dependencies directly on the problem representation. On the factorial representation, model-building (or search) is easy, while finding such a representation involves the difficult task of analyzing dependencies.

The general question we consider in this paper is how factorial representations can be found for arbitrary distributions. Since this is genuinely a machine learning question, we embed the discussion in machine learning concepts, especially latent variable models, which become in some sense a substitute for the notion of a (deterministic) genotype-phenotype mapping. This shift in notion does not lead to much more generality but it considerably simplifies the questions addressed in this paper and establishes insightful relations between problems like independent component analysis and the problem of learning genetic representations.

We will construct explicit factorial codes for certain kinds of distributions. One finds quickly that it is easy to construct *some* factorial representation for any distribution—in particular real-valued representations. But we will introduce specific (discrete-valued) codes that seem more promising as a basis for combinatorial search and GAs. Namely, we will introduce *Split Codings* and *Factorial Probabilistic Context-Free Grammars* (which are related to L-Systems).

The next section briefly introduces latent variable models. Sections 4 and 5 introduce the two mentioned codings. Section 6 gives an example. Section 7 gives a summary and

GECCO'05, June 25–29, 2005, Washington, DC, USA.

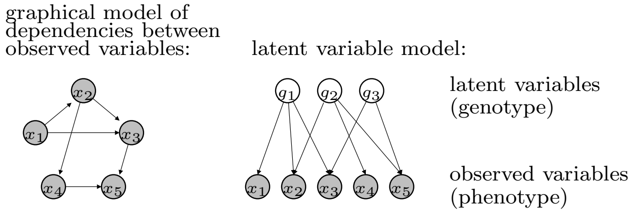
graphical model of dependencies between observed variables:
latent variable model:
latent variables (genotype)
observed variables (phenotype)
x_1
x_2
x_3
x_4
x_5
g_1
g_2
g_3

Figure 1: Direct graphical model vs. latent variable models to represent dependencies between variables.

discussion.

2. LATENT VARIABLE MODELS

The core problem of machine learning is to model data, e.g., to find a distribution that has high likelihood on the data. When the data is given in terms of several observable variables, fitting a graphical model to the data allows us to directly capture dependencies between them. Often though, a human would think that the dependencies between observables have their origin in unobservable processes.

Latent variable models follow this thinking: "latent variables are entities that we invent to explain patterns in the observed data" (from David MacKay's web page). Unfortunately it is true that mostly *we*, the human, invent these latent structures and not yet the algorithms themself. But the approach to fix the latent variable structure ad hoc and require the algorithm to fit this model to data is very powerful—a large number of algorithms can be derived in this way.

An important case is when the latent variables are restricted to be independent because fitting such a model to data means to understand the data as a superposition of patterns emitted from independent sources. Blind-source separation and Independent Component Analysis (ICA) are examples for this approach, and for the intimate relationship between "analyzing data" and "modeling data with a specifically structured latent variable model".

Our goal to find factorial representations for search fits perfectly in this picture. Observations of previously found good solutions reveal dependencies between their variables (*phenes*). What we try to find are independent latent variables (*genes*) that allow us to understand these dependencies as an effect of underlying hidden causes, while at the same time providing a representation on which these dependencies vanish and search decomposes. These latent variables very much play the same role as independent components in ICA.

Conventionally, the relation between phenotype and genotype is captured by a genotype-phenotype mapping ϕ. At first this seems a rather large shift in notion: the genotype-phenotype mapping deterministically associates a phenotype to each genotype while a latent variable model could have any probabilistic relation between them. At second sight though, this gain in generality is maybe not fundamental (see also next section). In fact, for the concrete codings that we develop (Split Coding and Factorial Grammars), there is a deterministic mapping from genotype to phenotype.

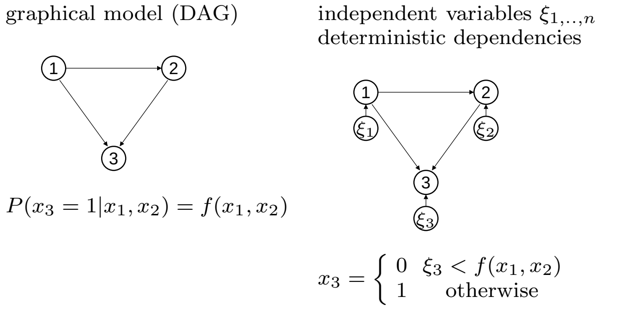
graphical model (DAG)
independent variables $\xi_{1,..,n}$ deterministic dependencies
1
2
3
ξ_1
ξ_2
ξ_3
$P(x_3 = 1|x_1, x_2) = f(x_1, x_2)$
$x_3 = \begin{cases} 0 & \xi_3 < f(x_1, x_2) \\ 1 & \text{otherwise} \end{cases}$

Figure 2: Any graphical model of n binary variables can trivially be written as a model with n independent latent variables $\xi_{1,..,n} \sim [0, 1]$ and only deterministic dependencies.

3. TRIVIAL FACTORIAL REPRESENTATIONS

Consider any probability distribution p over discrete variables. Clearly, one can write an algorithm that produces samples of this distribution. This algorithm will make use of a Random Number Generator (RNG). The random numbers drawn from the RNG are independent (idealistically they are continuous and uniform over $[0, 1]$). These random numbers are a latent factorial representation of the distribution and the algorithm represents a deterministic mapping from the latent variables to the observed ones.

This quite trivial observation is related to the question whether probabilistic dependencies between latent and observed variables are more general than deterministic: Figure 2 displays a graphical model of a distribution over 3 binary variables. The relation between all variables is probabilistic. One can though expand the graphical model, adding a real-valued random number $\xi \sim [0, 1]$ to each node. Then all dependencies between variables can be expressed deterministically, leaving the independent random numbers as the only probabilistic components.

These simple remarks give a first insight in factorial representations. However, it is questionable whether these kinds of representations (especially real-valued ones) have a practical relevance as a basis for genetic search. So we move on to more intuitive factorial representations for search.

4. SPLIT CODING

First, consider only two coupled binary variables x and y,

$$P(x, y) = P(x)\, P(y|x) \ .$$

What are possible factorial representations of this distributions? One way to think of this is that y is actually a combination of two underlying (hidden) variables, namely $y_0 \sim P(y|0)$ and $y_1 \sim P(y|1)$, see figure 3. The two hidden variables y_0 and y_1 are independent. In the underlying generative process, the three variables x, y_0 and y_1 are generated independently (based on their marginals). Then, depending on the value of x, y is assigned the same value as y_0 or y_1. In other terms, in an observed sample (x, y) only one of y_0 and y_1 is "expressed", the other one discarded, depending on the value of x.

The mapping from the three variables (x, y_0, y_1) to the observed variables (x, y) can be described in many ways with expression rules. A simple way is to do it with permutation rules. Such a rule is a triplet (i, j, k) which means "if

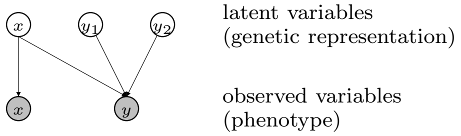

Figure 3: Split Coding for two binary variables x and y.

the ith variable has value 1, permute the jth with the kth variable". In our case, the permutation rules simply reads $(1, 2, 3)$, which means that if $x = 1$, y_0 and y_1 are permuted, leading to (x, y_1, y_0). After all permutations are done, the final result is given by clipping off the extra variables (leading to (x, y_1) for $x = 1$ and (x, y_0) for $x = 0$).

Consider a tree of n variables $x = (x_1, .., x_n)$,

$$P(x) = \prod_i P(x_i | x_{\pi(i)}) \ ,$$

where $\pi(i) \in \{1, .., i–1\}$ is the single parent node on which x_i is conditioned. Given such a tree, a factorial representation is constructed by *splitting* variables starting with the leafs, see figure 4. Splitting a variable x_j means to introduce a new variable x_k $(k = n + 1)$ with the marginal distribution $P(x_k) = P(x_j|1)$ while at the same time x_j is made independent from $x_{\pi(j)}$ by reassigning it the marginal distribution $P(x_j) \leftarrow P(x_j|0)$. This transformation is kept in memory by appending the permutation rule $(\pi(j), j, k)$ to the list of transformations.

If this is done with all variables in a tree, we will end up with $2n - 1$ independent variables (the root is not split) and a list of $n - 1$ permutation rules. Sampling can now be realized by sampling independent values for all of the $2n-1$ variables, applying the permutation rules in *inverse* order, and then picking only the first n variables.

4.1 Construction from data

An iterative scheme to construct a Split Coding is to first find a pair of variables with highest mutual information between them and then split one of the variables conditioned on the other one. This procedure will monotonously decrease the total mutual information[1].

However, the factorial representation constructed in that way typically introduces more latent variables than necessary because sometimes a variable is split which should, more efficiently, first be used as a condition of another variable. A more efficient order in which variables can be split is by a tree—which we call *Split-by-Tree Coding*:

Consider a population over n binary variables. The pairwise statistics for $P(x_i, x_j)$ and the empirical mutual information $I(i, j)$ are calculated for each i and j. Following [2] we construct a maximum spanning tree, maximizing the sum of mutual information associated to the edges. From that tree we construct the representation as explained in the previous section.

4.2 What does that buy us?

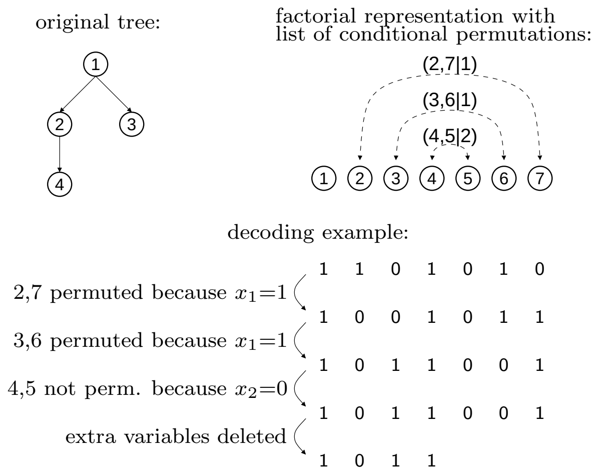
original tree:
1
2
3
4
factorial representation with
list of conditional permutations:
(2,7|1)
(3,6|1)
(4,5|2)
1
2
3
4
5
6
7
decoding example:
1 1 0 1 0 1 0
2,7 permuted because x_1=1
1 0 0 1 0 1 1
3,6 permuted because x_1=1
1 0 1 1 0 0 1
4,5 not perm. because x_2=0
1 0 1 1 0 0 1
extra variables deleted
1 0 1 1

Figure 4: Split Coding: A tree of 4 binary variables can be coded in 7 independent variables and a list of conditional permutation rules.

Modeling data by first constructing a Split-by-Tree Coding and then fitting a factorial distribution thereon is perfectly equivalent to modeling the data directly with a dependency tree. So one may ask: What have we gained?

First, of course, we have what we desired: a way to generate structured search distributions based on only factorial random variables (genes).

But there is an interesting point also from the machine learning perspective: *incrementality.* When the data is re-represented using such a coding, one can in principle apply any other modeling technique thereon. For instance, one can learn a dependency tree on the Split Coding—or repeat the Split Coding and learn a factorial model thereon. This opens new possibilities because if we had directly learned a dependency tree of the data, it is unclear how we could have further analyzed the data. In contrast, re-representing the data seems like removing all the structure from the data that we have already learned about in a first analysis, leaving all possibilities to continue with another analysis on that level.

Iterating the Split Coding also allows us to generate higher order dependencies. E.g., applying Split-by-Tree Coding twice on three binary variables constructs a representation on 9 binary variables which captures (if splitting is not done in an unlucky order) also all 3rd order (parity) interactions.

4.3 Is there another way to do it?

The Split Coding is a highly "redundant" encoding. Roughly speaking, the genome includes multiple partial solutions. Depending on another variable, either one or the other partial solution is expressed and copied into the phenotype. This clearly relates to the discussion of introns and the functional role of neutrality.

A legitimate question is whether there are not other, maybe more efficient ways to recode a solution in the simple tree case. Going back to the example with only two phenotypic variables, it is hard to think of another way to represent it more efficiently than by the three binary variables of the Split Coding. Eventually, every conditional probability $P(x_i|\cdot)$ in the direct representation must also be generated from the latent representation. If we restrict the latent variables to be discrete (and not the continuous random

[1] The worst case data for such kinds of models is the parity function: Consider a data set comprising all length n binary strings with positive parity. The mutual information is zero for any pair of variables, also for any triplet of variables, and even for any $n - 1$ tuple of variables. Only the nth order mutual information is non-zero (the same is true for the Walsh spectrum and Amari's θ coordinates).

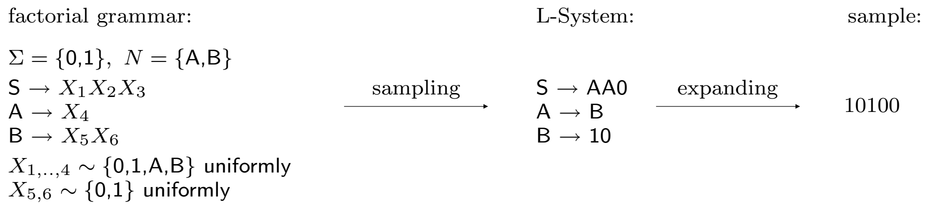

Figure 5: Example of a Factorial Probabilistic Context-Free Grammar (short: *Factorial Grammar*). Every RHS symbol $X_1, .., X_6$ is an independent random variable. Sampling a value for each RHS random variable we get an L-System which, by expression, gives the actual sample sequence.

numbers ξ of section 3) then the marginals of these latent variables somehow have to mimic the original conditionals.

5. FACTORIAL GRAMMARS (AND L-SYSTEMS)

Context-free grammars are defined by a start symbol S, a set Σ of terminal symbols, a set N of non-terminal symbols, and a set of productions $A \rightarrow \alpha$ where the left-hand-side (LHS) $A \in N$ and the right-hand-side (RHS) $\alpha \in (\Sigma \cup N)^*$. A language P is the set of all possible sequences that can be produced by a grammar. There are (typically) many different sequences in a language because there many productions that have the same LHS non-terminal symbol A and when producing a sample from the language one has free choice of choosing any of these productions to replace A with a RHS sequence.

While grammars define sets of sequences, probabilistic grammars define a distribution over sequences (see [5] for a good introduction). They have mainly been studied in the context of human speech and text models. In probabilistic grammars, a probability $P(A \rightarrow \alpha)$ is associated to each production. For fixed A, $P(A \rightarrow \alpha)$ is a normalized distribution and samples are generated from the grammar by choosing the productions according to this distribution.

The first question we might ask is: Can every distribution over sequences be represented by a probabilistic context-free grammar (PCFG)? The obvious answer is yes. In the extreme case for every possible solution x there exists a separate production $S \rightarrow x$ to which an arbitrary probability may be assigned. This is one PCFG that produces the desired distribution over P; and there are infinitely many equivalent ones, e.g., it could also be written in Chomsky normal form.

Going back to our basic question on how to construct factorial representations, there are two ways that PCFG guide. First, when taking samples from a PCFG, every decision on a production may be considered as an independent random variable. In that way, the factorial representation of a sample is a sequence of numbers that tell which production from the set of possible productions is to be taken during the sampling process. All of these numbers can be drawn independently and with marginal distributions equal to the probabilities $P(A \rightarrow \alpha)$ for fixed A. Unfortunately, we haven't made any further considerations or experiments yet with such representations. They seem interesting though.

A second approach is to consider a special class of PCFGs: First, since $P(A \rightarrow \alpha)$ is normalized for a fixed A, one may change the notation a bit and write $A \rightarrow P(\alpha)$, which means that a production maps a non-terminal symbol to a distribution over RHS sequences. In that view, a PCFG is equivalently given by a set of unique productions for every A, which all map to distributions over RHS sequences.

In a general PCFG, the distribution $P(\alpha)$ over the RHS may be arbitrary. Let us define *Factorial Probabilistic Context-Free Grammars* (in short, "*Factorial Grammar*") as those PCFGs where the distributions $P(\alpha)$ are constrained to factorize. In other terms, every RHS α is an arbitrary-length sequence of *independent* random variables (which have values in $\Sigma \cup N$). Drawing a sample for each of these random variables, the Factorial Grammar is turned into a specific context-free grammar where there is only a single unique production for every LHS symbol A. Such a grammar defines only a single sample (a language of cardinality 1) and can also be called *L-system* (or Lindenmayer-System [10]). See figure 5 for an illustration.

Now we may ask again: Can every distribution over P also be represented by a Factorial Grammar? Yes, and again there is a trivial but structurally not interesting way to construct such a Factorial Grammar. One could for every possible solution x have a separate non-terminal symbol A and a production $A \rightarrow x$ (which is deterministic). Then the start rule $S \rightarrow X$ can map the start symbol to a random variable X, where the probability of X having a value A is exactly the probability of x we want to represent. Eventually, this means to represent every possible solution by another symbol—which is indeed a factorial representation.

But there is a structurally more interesting way to construct a Factorial Grammar: First consider a tree as a latent variable graphical model of a set of observable binary variables (not to confuse with a tree over observable variables). The observables are the leaf nodes of the tree. Let the latent variables (non-leaf nodes) have values over a discrete and disjoint set of symbols. This latent variable model can easily be represented by a Factorial Grammar. The important thing to note is that the children of a node are *conditionally* independent given the value of the parent node. For instance, when somewhere in the tree three children variables X_2, X_3, X_4 depend on a parent node X_1 and the parent has values in $\{A, B\}$ then we may introduce two productions $A \rightarrow P(X_2|X_1 = A)P(X_3, X_1 = A)P(X_4|X_1 = A)$ and $B \rightarrow P(X_2|X_1 = B)P(X_3, X_1 = B)P(X_4|X_1 = B)$ to the set of productions of our Factorial Grammar. See figure 6 for an illustration.

On the other hand, every graphical model over binary variables can be represented by a latent variable tree: One first has to construct a junction tree from the graphical model and then take the junctions (not the cliques) as nodes of the latent variable tree. Here the structural complexity of the distribution plays the crucial role: if the graphical model was itself a tree, then it would directly serve as the

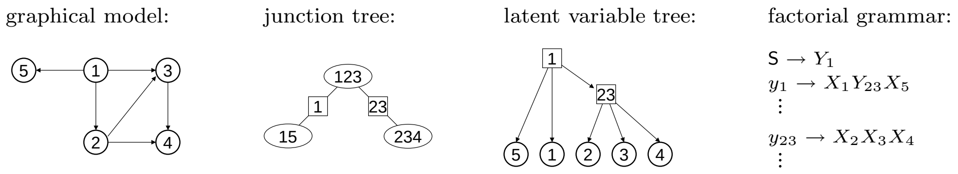

Figure 6: Taking the junctions of a (minimal) junction tree as latent variables of a tree representation, one can construct a Factorial Grammar that produces the same distribution as any graphical model (over discrete variables). The Factorial Grammar is abbreviated as follows: y_1 is a specific value of the latent variable Y_1 (which corresponds to the junction $\boxed{1}$) and for each such value there must be a different production rule (with different marginals over the RHS variables). The same is true for y_{23} and Y_{23}.

tree. If the graphical model is more complex, then the junction tree would comprise sets of variables as junctions. The junctions are the latent variables we were looking for: every junction is interpreted as a latent variable which emits the observable variables as leafs, see figure 6.

Intuitively, finding a *minimal* junction tree for the desired distribution and constructing the Factorial Grammar from that might result in a "structurally most concise" way to represent the distribution by a Factorial Grammar. But we don't have any proofs here.

5.1 How does that relate to conventional GAs?

The search mechanisms in standard evolutionary models, mutation and crossover, are of a factorial nature[2]. Accordingly, these mechanisms work very well on representations on which the problem factorizes.

All representations that we constructed in the previous sections can be used as underlying representations for a standard GA. For instance, consider the representation we constructed with the Factorial Grammar. Individuals of a GA would be instances of such a Factorial Grammar—i.e., L-Systems—where each symbol on the RHSs has a concrete value. The GA's population as a whole represents the Factorial Grammar in that it gives an (empirical) distribution over the RHS symbols. Actually, in such a GA, the distribution over the RHSs is not perfectly factorized (see footnote), but approximately. In particular, the mutation operator would induce independent variations of the RHS symbols and the crossover operator would annihilate linkages between these symbols. There are many examples for GAs where every individual represents an L-System (e.g., [7, 8, 15, 6, 3, 11]). None of them though constructs the grammar based on explicit data analysis; most of them are based on heuristic adaptations or self-adaptation of the L-Systems.

6. AN EXAMPLE

As an example, we calculate likelihoods of different models on data taken as follows. We considered a specific MAXSAT problem of the Satisfiability Library (www.satlib.org), namely the first of the length 50 Uniform Random-3-SAT problems (uf50-01.cnf). We generated 10 000 random sequences and on each one applied a standard Hill Climber that explores random bit alterations until no alteration leads to an improvement. In this way, we ended up with a set of 9974 different "local maxima". We take this set (associating equal weights to all of them) as the training data set, and we generate a second set (of size 9971) in the same way as test data.

We implemented three probabilistic modeling schemes:

Factorial Estimates the marginals $P(x_i)$ for each variable separately (cf. PBIL [1]); the distribution model is

$$P(x) = \prod_{i=1}^{n} P(x_i) \ . \qquad (1)$$

Tree First calculates the mutual information between each pair of variables; then generates a maximum spanning tree with random root, maximizing the total mutual information associated to the edges (cf. COMIT [2]). The distribution model is

$$P(x) = P(x_0) \prod_{i=2}^{n} P(x_i | x_{\pi(i)}) \ , \qquad (2)$$

where $\pi(i) \in \{1, .., i-1\}$ is the parent of i and the indexing of variables is topologically sorted w.r.t. the tree.

3rd order graphical model First calculates the mutual information between each *triplet* of variables ($I(i,j,k) = H(i) + H(j) + H(k) - H(i,j,k)$). Then generates a graph, where each node (except the first two) has exactly two parents. This graph is generated following the same scheme as for the maximum spanning tree: At every step, find a variable x_k that has not yet been added to the graph and *two* nodes x_i and x_j that have already been added, which maximize $I(i,j,k) - I(i,j)$; then add x_k to the graph with edges from x_i and x_j. The distribution model is

$$P(x) = P(x_0) \, P(x_1|x_0) \prod_{i=3}^{n} P(x_i | x_{\pi_1(i)}, x_{\pi_2(i)}) \ , \quad (3)$$

[2] GAs are heuristic search schemes, and as such they can be understood by investigating what kind of search distribution they induce depending on previously found good solutions (the parent population). A traditional GA has independent symbol alterations as a mutation operator, which induces a factorized variability distribution σ for each individual. Considering the whole parent population, a mutation only GA has a mixture of factorized distributions as search distribution. Crossover is an operator that makes this search distribution "more factorized". Actually, one can understand crossover as a move in distribution space from the parent population to a more factorized version of the parent population without changing the gene marginals. One can show that both, mutation and crossover, can only reduce the total mutual information that was present in the parent population [13]. In any case, although GAs do not induce a perfectly factorized search distribution, its inherent search mechanisms are of a factorial nature. This is also the reason why they have been modeled as factorial search, e.g., by PBIL or gene pool GAs.

probabilistic model	log-likelihood on training data	log-likelihood on test data
tree model	-42.0819	-42.2157
3rd order graphical model	-38.8017	-38.9608
factorial on direct representation	-46.0961	-46.1495
factorial after 1 Split-by-Tree Coding	-42.0819	-42.2157
factorial after 2 Split-by-Tree Codings	-39.7242	-39.9237

Table 1: The log-likelihood of different models for a distribution derived from a MAXSAT problem. (A uniform distribution over sequences of length 50 had a log-likelihood of -50; a mixture of 10 000 delta-distributions for each of 10 000 training data points had a log-likelihood of -13.3 (and $-\infty$ on test data).)

where $\pi_1(i) \neq \pi_2(i) \in \{1, .., i-1\}$ are the two parents of i (node indexing is topologically sorted w.r.t. the graph). The root x_0 is chosen randomly and the second node x_1 as for the tree building.

We also test the Split-by-Tree Coding by calculating the likelihood of the factorial model on different levels of re-representation. In more detail: We calculate the coding from the data, re-represent the data, model it with the factorial model, and calculate the likelihood of the data w.r.t. this factorial model on that latent representation. Note that "calculating the data likelihood on the latent representation" is non-trivial since one has to eliminate over the latent variables that are not expressed—but with the factorial model this is straight-forward to do. We iterate this procedure twice: calculating a new Split Coding of the re-represented data and fitting a factorial model thereon.

Table 1 displays the results. As a sanity check we find that the factorial model after 1 Split-by-Tree Coding is equivalent to the tree model. The result of the factorial model becomes significantly better after another iteration of the Split Coding. Note that for the 3rd order graphical model, one has to calculate statistics for $\binom{n}{3}$ triplets of variables (which dominates the algorithm complexity with $O(D\, n^3)$, when D is the size of the data set), whereas for the Split-by-Tree Coding one has to calculate $\binom{n}{2}$ pair statistics in the 1st iteration, and $\binom{2n-1}{2}$ in the 2nd iteration (which is $O(D\, n^2)$).

Constructing an EDA from these ideas is easy; the factorial representation is used only for the purpose of modeling a distribution. E.g., using the Split-by-Tree Coding and a factorial distribution model thereon, the algorithm is exactly equivalent a direct EDA with the tree model.

Constructing a GA from these ideas though is not easy. A straight-forward approach might follow the technique given in [14]: Given a parent population, construct a factorial representation from its statistics, e.g., with the Split-by-Tree Coding. Then map all parents onto this factorial representation. Then apply crossover and mutation on this representation to generate an offspring. Then map this offspring back on the problem representation and evaluate.

An interesting problem though arises when mapping the parents to the factorial representation. In [14], the compressed representation of a parent was uniquely defined once the compression map was fixed. But in the Split Coding, there are many genotypic representations that map to the same phenotype. In the sketched algorithm it is quite unclear which genetic representation to choose for a given parent. In preliminary experiments with the Split-by-Tree Coding, we sampled the undetermined genetic variables (one might call them neural genes or introns) from its distribution in those parents where it was determined (which is a bit like an E-step, inferring non-observed variables). However, we found only statistically non-significant improvements of such a GA (with standard mutation and uniform crossover) on a length 100 MAXSAT problem when compared to a direct representation GA.

In fact, it seems inefficient that in every generation an individual first looses all information about its latent genetic variables when it is mapped on the problem representation, and then, when it is mapped back on a factorial representation, heuristics have to reassign values to the undetermined genetic variables. If the mapping (the set of conditional permutation rules) were non-adaptive, the values of the latent variables could of course be memorized for each individual (be part of the genotype for a fixed genotype-phenotype mapping). But when the mapping is recalculated (or adapted) in every generation, it is unclear how to preserve such information.

The most *natural* solution to such problems is the self-adaptive evolution of genetic representations ([11] gives an example for L-Systems), where the description of an individual includes all latent variables as well as how they have to be expressed. But this does not accord with our aim to explicitly construct a factorial representation from the statistics of previously evaluated samples. Future research will have to address these issues.

7. DISCUSSION

Let us summarize the factorial representations we addressed in this paper:

(1) First, for any distribution, if we have an algorithm producing samples of the distribution, we can consider the random numbers of the algorithm's RNG as a factorial representation. But it is questionable whether this representation is promising for search.

(2) Every distribution can be represented by a Probabilistic Context-Free Grammar. Related to (1), we can consider the sequence of decisions made when sampling from such a grammar as a factorial representation. This seems interesting to consider as a basis for search, but the implications are yet really unclear.

(3) Every distribution can also be represented by a Factorial Grammar, where the symbols on each RHS of a production are independent variables. This is a very plausible and common representation for genetic search—an L-System is a sample from a Factorial Grammar and there exist many GAs based on L-Systems.

(4) For distributions that can be modeled by a dependency

tree we can efficiently construct a Split Coding as a factorial representation. For arbitrary distributions, iterating the Split Coding ensures that the mutual information can only decrease (we did not prove convergence to a perfect factorial representation). The Split Coding seems to be a practical approach to handle interactions between arbitrary, non-contiguous variables.

(5) At another place [14] we have shown how compact genetic codes are related to factorial representations.

From a pure machine learning perspective, a potential benefit of modeling distributions by re-representation of the data is that it allows us to apply any additional analysis technique on that representation. An example is the iterated Split-by-Tree Coding, which is computationally cheaper than a direct 3rd oder analysis. Maybe we also contributed a bit to the question of how latent variables that explain patterns in observed data can be discovered automatically instead of being invented by humans.

Finally, we think there are also implications from a biological perspective. We showed at another place [12] that a standard evolutionary process induces an implicit selection pressure on genetic representations—favoring representations that generate a phenotypic variability that fits with the Boltzmann fitness distribution. Here we showed that factorial genetic representations are sufficient to generate any desired distribution. Although we would not claim that natural genetic systems are, e.g., comparable to a Factorial Grammar, we do claim that the mechanisms involved in natural genetic systems are at least as rich as those of a Factorial Grammar. A tempting conclusion is that genes are factorial latent variables invented by nature to generate arbitrary search distributions.

Acknowledgment

The author would like to thank Chris Williams for helpful discussions on the topic of latent variable models, and the German Research Foundation (DFG) for their funding of the Emmy Noether fellowship TO 409/1-1, allowing me to pursue this research.

References

[1] S. Baluja. Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. Technical Report CMU-CS-94-163, Comp. Sci. Dep., Carnegie Mellon U., 1994.

[2] S. Baluja and S. Davies. Using optimal dependency-trees for combinatorial optimization: Learning the structure of the search space. In *Proc. of Fourteenth Int. Conf. on Machine Learning (ICML 1997)*, pages 30–38, 1997.

[3] E. D. de Jong. Representation development from Pareto-Coevolution. In *Genetic and Evolutionary Computation Conference (GECCO 2003)*, 2003.

[4] E. D. de Jong, R. A. Watson, and D. Thierens. On the complexity of hierarchical problem solving. In *Genetic and Evolutionary Computation Conference (GECCO 2005)*, 2005.

[5] S. Geman and M. Johnson. Probabilistic grammars and their applications. DRAFT of October 24, 2000; available at http://www.ima.umn.edu/talks/workshops/10-30-11-3.2000/johnson/grammars.pdf.

[6] G. S. Hornby. Generative representations for evolutionary design automation, 2003. Ph.D. Dissertation, Brandeis University Dept. of Computer Science.

[7] G. S. Hornby and J. B. Pollack. The advantages of generative grammatical encodings for physical design. In *Proc. of 2001 Congress on Evolutionary Computation (CEC 2001)*, pages 600–607. IEEE Press, 2001.

[8] G. S. Hornby and J. B. Pollack. Evolving L-systems to generate virtual creatures. *Computers and Graphics*, 25:1041–1048, 2001.

[9] M. Pelikan, D. E. Goldberg, and F. Lobo. A survey of optimization by building and using probabilistic models. Technical Report IlliGAL-99018, Illinois Genetic Algorithms Laboratory, 1999.

[10] P. Prusinkiewicz and J. Hanan. *Lindenmayer Systems, Fractals, and Plants*, volume 79 of *Lecture Notes in Biomathematics*. Springer, New York, 1989.

[11] M. Toussaint. Demonstrating the evolution of complex genetic representations: An evolution of artificial plants. In *Genetic and Evolutionary Computation Conference (GECCO 2003)*, pages 86–97, 2003.

[12] M. Toussaint. On the evolution of phenotypic exploration distributions. In C. Cotta, K. De Jong, R. Poli, and J. Rowe, editors, *Foundations of Genetic Algorithms 7 (FOGA VII)*, pages 169–182. Morgan Kaufmann, 2003.

[13] M. Toussaint. The structure of evolutionary exploration: On crossover, buildings blocks, and Estimation-Of-Distribution algorithms. In *Genetic and Evolutionary Computation Conference (GECCO 2003)*, pages 1444–1456, 2003.

[14] M. Toussaint. Compact genetic codes as a search strategy of evolutionary processes. In *Foundations of Genetic Algorithms 8 (FOGA VIII)*, LNCS. Springer, 2005.

[15] R. Watson and J. Pollack. A computational model of symbiotic composition in evolutionary transitions. *Biosystems, Special Issue on Evolvability*, 69:187–209, 2002.